# Java Collection Framework

CSC 209 Data Structures

**Kulwadee Somboonviwat**

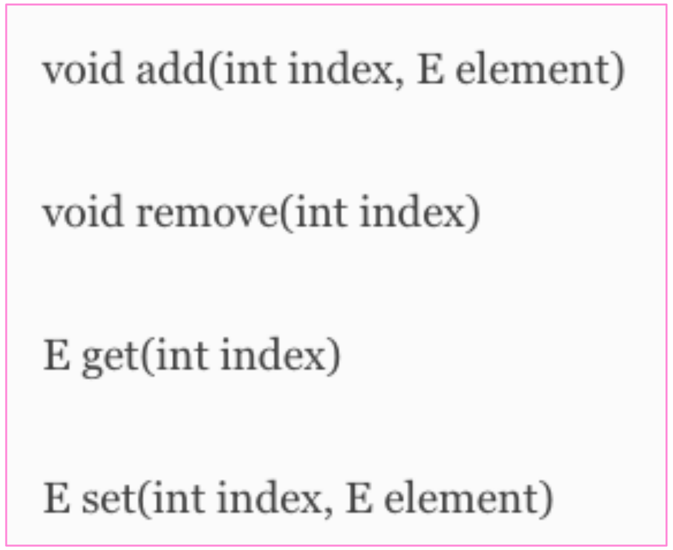
kulwadee.som [at] sit.kmutt.ac.th

# Java Collection Framework

- A **collection** is an object that represents a group of objects (e.g. Vector, List, Queue)
- Java Collection Framework (JCF)
  - A unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details
  - Separate interfaces and implementations

## The List Interface

- **List** is an *ordered* collection.
- Elements are added into a particular position in the List collection.
- Interface Methods:



```
void add(int index, E element)

void remove(int index)

E get(int index)

E set(int index, E element)
```

For more detail, check the documentation of java.util.List at

https://docs.oracle.com/javase/7/docs/api/java/util/List.html

## Concrete Implementations

- **ArrayList.** resizable array
- **LinkedList.** doubly-linked list

Some operations are faster or use less space with **ArrayList**; others are faster or smaller with **LinkedList**. Which one is better for a particular application depends on which operations it performs most often.

For more detail, check the documentation of java.util.List at

https://docs.oracle.com/javase/7/docs/api/java/util/List.html

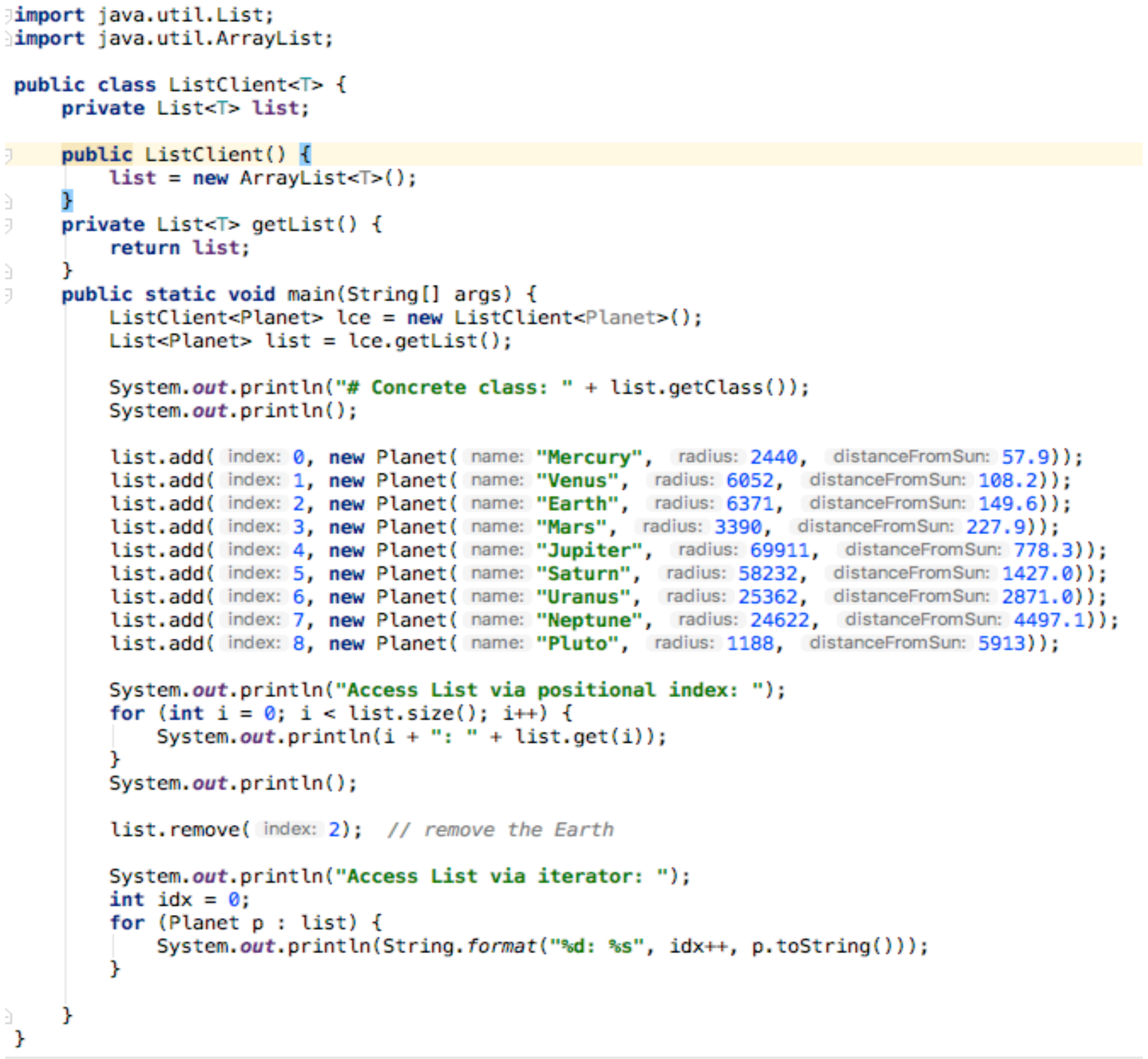
```
public class ListClientExample {
    private List list;

    public ListClientExample() {
        list = new LinkedList();
    }

    private List getList() {
        return list;
    }

    public static void main(String[] args) {
        ListClientExample lce = new ListClientExample();
        List list = lce.getList();
        System.out.println(list);
    }
}
```

If you later find out that ArrayList can give better performance for your application than LinkedList, you can just replace **new LinkedList()** with **new ArrayList**



```
import java.util.List;
import java.util.ArrayList;

public class ListClient<T> {
    private List<T> list;

    public ListClient() {
        list = new ArrayList<T>();
    }
    private List<T> getList() {
        return list;
    }
    public static void main(String[] args) {
        ListClient<Planet> lce = new ListClient<Planet>();
        List<Planet> list = lce.getList();

        System.out.println("# Concrete class: " + list.getClass());
        System.out.println();

        list.add( index: 0, new Planet( name: "Mercury",  radius: 2440,  distanceFromSun: 57.9));
        list.add( index: 1, new Planet( name: "Venus",  radius: 6052,  distanceFromSun: 108.2));
        list.add( index: 2, new Planet( name: "Earth",  radius: 6371,  distanceFromSun: 149.6));
        list.add( index: 3, new Planet( name: "Mars",  radius: 3390,  distanceFromSun: 227.9));
        list.add( index: 4, new Planet( name: "Jupiter",  radius: 69911,  distanceFromSun: 778.3));
        list.add( index: 5, new Planet( name: "Saturn",  radius: 58232,  distanceFromSun: 1427.0));
        list.add( index: 6, new Planet( name: "Uranus",  radius: 25362,  distanceFromSun: 2871.0));
        list.add( index: 7, new Planet( name: "Neptune",  radius: 24622,  distanceFromSun: 4497.1));
        list.add( index: 8, new Planet( name: "Pluto",  radius: 1188,  distanceFromSun: 5913));

        System.out.println("Access List via positional index: ");
        for (int i = 0; i < list.size(); i++) {
            System.out.println(i + ": " + list.get(i));
        }
        System.out.println();

        list.remove( index: 2);  // remove the Earth

        System.out.println("Access List via iterator: ");
        int idx = 0;
        for (Planet p : list) {
            System.out.println(String.format("%d: %s", idx++, p.toString()));
        }

    }
}
```

## Practice Exercise

- Change the concrete class of the list object of the **ListClient** class from **ArrayList** to **LinkedList**

Source code available at the course’s GitHub Repository:
https://github.com/kulwadeesom/csc209_256002

Class **co.kulwadee.csc209.lect02. ListClient**